

Mobile Apps Development 7



Getting to know your unit

Assessment

You will be assessed by a series of assignments set by your tutor.

Mobile devices are everywhere and their impressive sales are primarily driven by the innovative apps that keep their users educated, informed and entertained. The mobile applications sector enjoys a thriving commercial ecosystem and it can be profitably targeted by small start-ups and larger corporations alike.

Developing a mobile application is typically the result of having a good initial idea or solving an essential problem; making that mobile application really successful comes from an appreciation of the intricacies of mobile devices, the various forms of functionality available and how apps are designed to be intuitively usable. The ability to create such applications is a highly sought skill and will help you, as a software developer, gain a competitive edge.

How you will be assessed

This unit will be assessed internally by two tasks set by your tutor. Throughout this unit, you will find assessment practice activities that will help you work towards your assessments. Completing these activities will not mean that you have achieved a particular grade, but you will have carried out useful research or preparation that will be relevant when it comes to your final assignment.

In order for you to complete the tasks in your assignments successfully, it is important to check that you have met all of the Pass grading criteria. You can do this as you work your way through the assignments.

If you are hoping to gain a Merit or Distinction, you should also make sure that you present the information in your assignments in the style that is required by the relevant assessment criteria. For example, Merit criteria require you to review and justify and Distinction criteria require you to evaluate.

The assignments set by your tutor will consist of a number of tasks designed to meet the criteria in the table. The first assignment is likely to consist primarily of a research-based written task that requires you to investigate mobile apps and mobile devices, while the second will include practical activities such as:

- ▶ designing a mobile app that utilises device functions
- ▶ developing a mobile app that utilises device functions.

Assessment criteria

This table shows what you must do in order to achieve a **Pass**, **Merit** or **Distinction** grade, and where you can find activities to help you.

Pass	Merit	Distinction
Learning aim A Investigate mobile apps and mobile devices		
A.P1 Explain how the purpose of a mobile app and the needs, preferences and characteristics of the user affect its design and the provided features. Assessment practice 7.1	A.M1 Analyse how the implementation and design of mobile apps is affected by the intended user, current technologies and the purpose of the app. Assessment practice 7.1	A.D1 Evaluate how the effectiveness of mobile app implementation and design are affected by the intended user, current technologies and the purpose of the app. Assessment practice 7.1
A.P2 Explain the impact of current technologies on the design and implementation of mobile apps. Assessment practice 7.1		
Learning aim B Design a mobile app that utilises device functions		
B.P3 Produce designs for a mobile app to meet identified requirements. Assessment practice 7.2	B.M2 Justify how decisions made during the design process ensure the design for the app will meet identified requirements. Assessment practice 7.2	BC.D2 Evaluate the design and optimised mobile app against client requirements. Assessment practice 7.2
B.P4 Review the mobile app designs with others to identify and inform refinements. Assessment practice 7.2		
Learning aim C Develop a mobile app that utilises device functions		
C.P5 Produce a mobile app that meets the design criteria. Assessment practice 7.2	C.M3 Optimise a mobile app that meets the design criteria. Assessment practice 7.2	BC.D3 Demonstrate individual responsibility, creativity and effective self-management in the design, development and review of a mobile app. Assessment practice 7.2
C.P6 Test a mobile app for functionality, usability, stability and performance. Assessment practice 7.2		
C.P7 Review the extent to which the mobile app meets the identified requirements. Assessment practice 7.2		

Getting Started

Developing mobile applications requires the knowledge and application of many different skills. Write down a list of the tools and techniques you think you will need in order to create innovative mobile applications. When you have completed this unit, see if you have missed any obvious skills, tools or technologies from this list.



A Investigate mobile apps and mobile devices

Before starting to develop mobile applications, it is important to gain an appreciation of the range of devices and functionality available. Many design decisions are shaped by the freedoms and constraints of the target devices. In this section, we will investigate different types of mobile app, the devices on which they work and the features, characteristics and options that will shape your development path.

Types of mobile apps

Mobile applications (more commonly known as mobile apps) are programs designed to work on smartphones, tablet PCs and emerging wearable technology such as watches and glasses. An app is typically developed as one of three general types outlined in Table 7.1.

Native and hybrid applications require specialist software (and sometimes hardware) in order to be developed successfully. For example, Apple® iPad® and iPhone® apps are normally created using Apple®-specific software and hardware which guarantees development standards and practices.

In direct contrast, web applications can be developed using simple text-editor tools such as Microsoft® Notepad and may work equally well from within web browser client software running on a desktop PC, games console or

mobile device. By being so versatile, they reward greater investment (time, money etc.) by developers because the potential user base is much larger.

Generally speaking, when choosing which type of mobile app to develop, there is a trade-off between development time and cost, the required functionality and the level of user experience required.

Discussion

If you have a mobile device, you are likely to have encountered a wide variety of different mobile applications. Think about these apps, especially their appearance and functionally, and determine whether they are likely to be native, web or hybrid apps. Discuss your conclusions with your class.

Context of mobile apps

Mobile apps are created to fulfil many different user needs, and this defines their context and purpose. Each app is designed to provide the user with a specific tool, experience or enhancement that their basic mobile device does not already provide (or do well). The overall interface design, development challenges and actual use once installed on the mobile device is dependent on the type of app.

► **Table 7.1:** The three types of mobile app

Native apps	Web apps	Hybrid apps
<ul style="list-style-type: none">• Programmed for, and installed on, a specific mobile platform.• Can take advantage of all (permitted) functionality available in the device, eg location, camera, contacts.• Typically mimic manufacturer's 'official' apps, drawing on the user's prior experience for ease of use.	<ul style="list-style-type: none">• Remote applications, typically running on a server.• User typically interacts with the application through a mobile device's web browser.• No application or data is typically installed on the device.• Access to device functionality is limited.• May visually mimic the device's usual interface.	<ul style="list-style-type: none">• Usually cross-platform compatible.• Typically uses website components that are 'wrapped' inside a native application.• Access to device functionality may be limited.• May be seen as a more cost-effective development approach, especially when multiple types of mobile device are being targeted.

► **Table 7.2:** Common categories of app context and purpose

Context	Purpose and features	Examples
Locale	Apps designed to provide you with geographical information based on their current location (eg maps, GPS route finders, augmented reality (AR) experiences).	Google Maps™, Wikitude™, Yelp Monocle™, Google™ Sky, Aurasma™
Utility	Apps provided to help configure or maintain your device (eg file manager (adds or removes files), backup tools (backup to the cloud), system monitors).	Glary Utilities™, CCleaner™, WinZip™, AVG® AntiVirus for Android™
Productivity suites	Apps giving office-style functionality for word-processing, spreadsheets, slideshows, databases. They provide the user with facilities to write letters or reports, create presentations, calculate project costings etc.	Adobe® Acrobat®, Microsoft® Office®, Office 365® and OneNote®, Dropbox™
Immersive full screen	Apps using the screen display exclusively to fully draw you into their experience (eg games).	Temple Run 2™, Candy Crush Saga™, Angry Birds™, Fruit Ninja™, Minecraft: Pocket Edition™
Lifestyle	Apps designed to enrich your life (eg healthy recipes, DIY, interior design, gardening, travel). They may provide step-by-step visual instructions and narrative, 3D mock-ups and links to commercial websites.	Craftsy™, Gumtree™, TripAdvisor®, Etsy™, Evernote™, Google Translate™, eBay™
Social	Apps designed to connect you to other people, to aid communication and share ideas, events, pictures and videos.	Twitter™, Facebook™, Snapchat™, WhatsApp™, Pinterest™, Instagram™, WordPress™, Blogger™
Entertainment	Apps that provide media content (eg music players, video streaming, podcasts, e-book readers).	YouTube™, BBC iPlayer™, Spotify®, Shazam™, SoundCloud™, Kindle™, Audible™
Widgets	Apps that take up very little of the mobile device's screen display which provide quick access to live data or settings (eg news tickers, quick device settings, search facilities, calendar and appointments).	Feedly™, Flashlight apps, Todoist™, BBC News

Building apps for each of these categories poses different challenges to the mobile app developer.

In simple terms, the mobile app you design and its features will be determined by the task(s) that it must perform and the personal preferences and needs of its targeted user.

Identifying specific user needs can be difficult, but there are some key points and questions to remember.

- User needs – what functionality does the user need from the app? Which features are required?
- User preferences – how does the user want to use the app? How does the user want to interact with the app? How does the user want to navigate the app's functions and features? What visual style should the app offer?
- User characteristics – how should the app cater for different types of user, eg age of the user, technical expertise of the user, disabilities such as visual impairment or hearing difficulties, the physical environment in which the app is used, the breadth of user configuration that the app offers, the level of help and assistance the app provides?

Answering these types of question builds a profile of the app's intended users and helps to shape both its design and included features.

You have probably experienced occasions where you have downloaded two similar mobile apps that claim to do the same job; their design, effectiveness and usability will help you evaluate which to keep and which to remove.

Theory into practice

Podcasting has become a popular form of entertainment in the last few years and there are many different podcast apps available for the popular mobile platforms.

Select three different podcast apps for a selected mobile platform and investigate their design, functions, features, performance and usability.

Which one would you recommend to a friend? Explain and justify your reasoning.

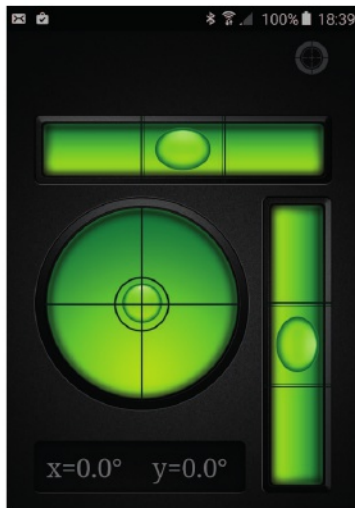
Mobile device integration

Mobile devices come in many different shapes and sizes (smartphones, tablets, wearable technology). In addition, their features and functions also vary greatly. This is partly due to the fact that mobile manufacturers wish to distinguish their products in the marketplace and the integration of specific sensors (fingerprint security,

for example) can be an attractive selling point. When designing a mobile app, it is necessary to be aware of these different characteristics and the implications they may have for both design and development.

Mobile device functions

As a developer, you cannot guarantee that a mobile device has a particular function when designing and developing an application so it is recommended that you check the operating system to see if the function is available. However, key functions such as vibration, headphone and speaker output, touch screens, microphone and still and video cameras can generally be guaranteed across any modern mobile device, irrespective of the manufacturer.



► **Figure 7.1:** Spirit Level mobile app using an orientation sensor

► **Table 7.3:** Common mobile device functions, what they are and how they are typically used

Mobile device function	What it is	How it is used
Accelerometer	A motion sensor measuring the force applied to the device on all three axes (x, y and z), excluding gravity.	Used to track motion such as shaking the device and tracking the difference between a user walking or jogging.
Magnetometer	A positional sensor measuring the ambient magnetic field.	Used to create a compass.
Thermometer	An environment sensor measuring ambient or internal temperature, typically in degrees Celsius (°C).	Used to measure room temperature or the temperature of the mobile device itself.
Barometer	An environment sensor measuring air pressure usually in either hectoPascal (hPa) or millibars (mbar).	Used to measure air pressure in a room. Also used to measure altitude, to make GPS more accurate when height is important.
Photometer	An environment sensor measuring the ambient light levels in lux (lx).	Used to auto-adjust screen brightness depending on external light conditions to improve readability and conserve power.
Orientation	A motion sensor calculating tilt and orientation (see Figure 7.1). This uses the accelerometer.	Used to change the orientation of an app's display when a device is changed from being held in a portrait to landscape manner, or to create a spirit level.
Global positioning system (GPS)	A positional sensor using triangulating satellites or cellular base stations to calculate a real-world location.	Used to track the physical location of the mobile device for providing location-sensitive information (eg nearest restaurant or petrol station) or services (finding friends), live maps and route finding.

User interface

A user interface (often abbreviated to UI), is the combination of software and hardware that a user interacts with to perform set tasks. The software part of the UI includes the underlying operating system and the selected mobile app itself. The hardware of the UI is represented by input devices (such as touch screens, physical buttons, camera and microphone) and output devices (such as screen, speaker and vibration).

Mobile apps development is challenging because there are many limitations that the designer and programmer need to navigate successfully in order to create a rewarding UI interactive experience. These challenging characteristics include:

- limited (or potentially variable) screen size
- limited keyboard/keypad input mechanism
- limited processing power (although in modern smartphone devices this has rapidly improved).

The use of alternative input mechanisms such as voice, touch control, complex gestures (pinch, stretch and swipe) and physical effects (shake and tilt) require extensive thought before they are used in an app, in order to produce a user-friendly experience. When used well, for example in a sketching app that allows the user to shake the device to wipe the picture clean, the results are pleasing and the experience is highly intuitive. When designed poorly, for example controlling a nimble playable character using poorly judged touch screen controls, the results can be imprecise and frustrating for users.

Even devices in the same family (for example Apple® iPhone® or iPad®) may have slightly different UI options, so you should never really take any feature for granted without researching it properly.

Variable features may include the type of touch screen (**resistive** or **capacitive**-based, single or multi-touch), dedicated physical buttons or just a very small screen. Mobile app developers need to think about how their application will function within the constraints of different user interfaces (and on different hardware) in order to support their targeted users effectively.

Key terms

Resistive touch screens – use two layers (usually glass and plastic) covered in an electrical conductive material (usually Indium Tin Oxide (ITO)). The two layers are kept apart until a finger or stylus presses them together, which causes a localised change in electrical resistance. This type of touch screen is cheaper to manufacture than capacitive touch screens but is not very sensitive and cannot support multi-touch gestures.

Capacitive touch screens – use two spaced layers of glass that are both coated with minute ITO capacitors. When the user's finger touches the screen, it changes the screen's local electrostatic field. This type of touch screen is brighter and more sensitive than resistive touch screens, and supports multi-touch gestures. However, the structural complexity of the screen raises its production costs.

Operating system

Mobile devices are controlled by their operating systems (OS) in a similar fashion to notebooks and PCs. Similarly, just as notebooks and PCs offer different operating systems (Microsoft® Windows®, Linux™, Apple® OS X®), the majority of the mobile device marketplace is divided between those devices running the Google™ Android™ OS (see Figure 7.3) and those using Apple® iOS (see Figure 7.2).

Both mobile operating systems support similar functionality but the way in which a developer makes use of each function as their app runs is often quite different, and this can also vary between versions of a mobile device OS. For example, it is quite common for a new OS version to 'break' a mobile app that was previously working because it has changed some small detail or setting.

In addition, the programming language used to build the app for each OS is typically different and provides its own challenges, rewards, advantages and disadvantages.

Traditionally Android™ apps are written using Oracle®'s Java™ (although C or C++ may also be used), while Apple® iOS apps are created using Objective-C.



► Figure 7.2: Apple® iOS



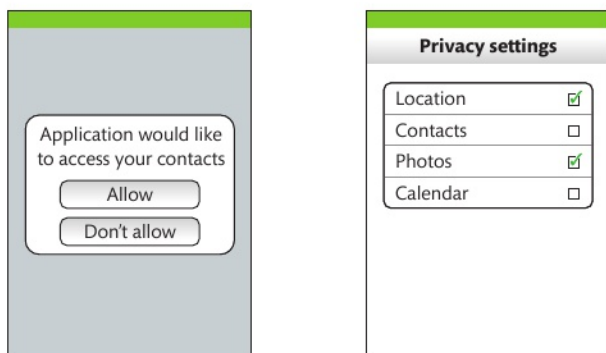
► Figure 7.3: Google™ Android™

Device permissions

Mobile app developers will often want to access particular data or functionality contained in a mobile device in order to make their apps more appealing to customers. Mobile device manufacturers take security and the integrity of their devices very seriously; therefore, apps often need to be granted permission to use data and functionality within a device through manual user intervention.

You may well have experienced this yourself. A pop-up prompt will appear asking if the mobile app you are using may access your location, read your contacts or phone status, and even access your network (see Figure 7.4). Some mobile apps, such as podcast players, may even need permission to download data if not connected to a wireless network, simply because mobile network data charges can be expensive.

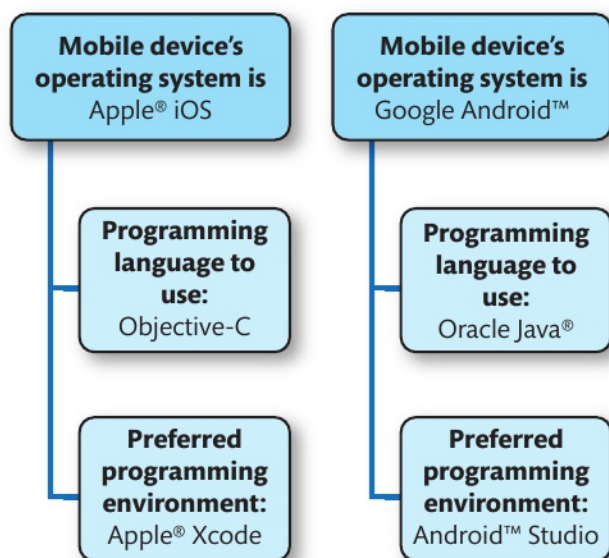
As a mobile app developer, you should not plan for your app to rely on data or functionality that might not be granted by the user.



► **Figure 7.4:** An operating system asking permission from the user for the app to access their Contacts information

Mobile app programming

As a mobile app developer your choices, including the programming language you must learn and the integrated development environment in which you build your application, are typically predetermined by the OS of the mobile device for which you intend to build your app, as shown in Figure 7.5.



► **Figure 7.5:** Determining the development tools and programming language

Programming languages

Java® is an established and popular programming language, which is over 20 years old and is used in billions of devices worldwide. The Android™ **software development kit (SDK)** uses Java® as the basis for building its mobile apps. Objective-C, a superset of an older language called C, has been around since 1983 and is the main language used by Apple® for its desktop OS X® and mobile iOS operating systems.

Despite being different programming languages, Objective-C and Java® are both **object-oriented programming languages** and are similar, in terms of their structure and syntax. As a result, you may find that becoming confident in one will be beneficial for learning the other.

Key terms

Software development kit (SDK) – a suite of software tools provided for developers to create an application for a specific platform, for example Java® SDK for Android™. The SDK usually contains application programming interfaces (APIs), tools such as compilers and debuggers, reference documentation and code samples.

Object-oriented programming languages – a modern programming approach (paradigm) to software development that works by modelling real-world problems and simplifies complex processes into the basic interactions that exist between different objects: for example, a customer and their bank account.

Developing for Android™ is essentially free, although a one-off \$25 registration fee is required for publishing free or commercial apps that are to be distributed through Google Play™. Apple® iOS development appears to be more expensive as it requires a \$99 registration fee annually.

Research

Visit Apple®'s membership support page and find out about the various membership options that enable development of iOS apps for their mobile devices. To access this website go to: developer.apple.com

Learn how to register and publish Android™ apps through Google Play™ by visiting: developer.android.com

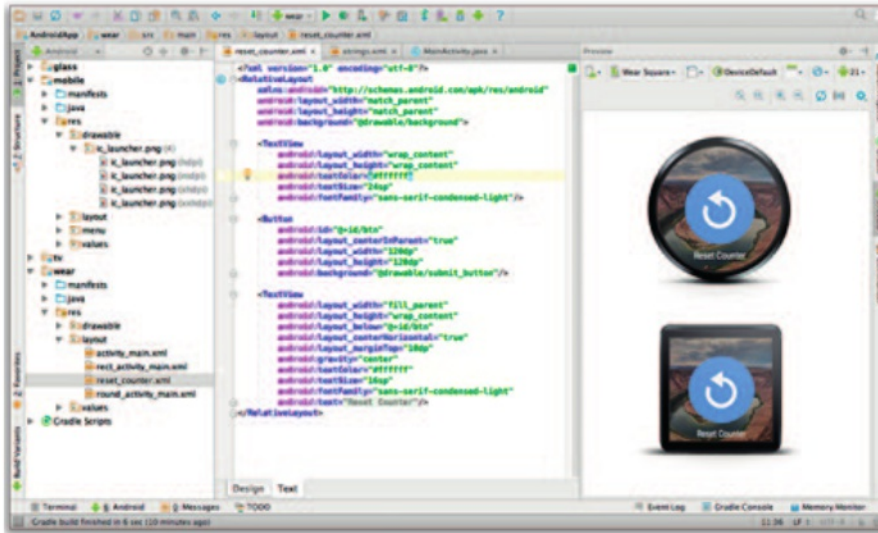
This book focuses primarily on Android™ and Java®, but you will find that doing research to make some comparisons is very useful.

Programming environments

The role of the modern programming environment is represented by an integrated development environment (IDE). The IDE provides developers with a comprehensive suite of tools, typically including:

- project management tools – to create, organise and manage your apps
- design tools – to create user interfaces, typically using 'drag and drop' techniques
- fully featured text editor – to key-in, edit and save your app's source code
- code completion – helpful auto-complete to speed-up development

- ▶ compiler – to translate your app's source code so it works on its target mobile device
- ▶ syntax highlighting – colouring different elements of program code to improve readability
- ▶ documentation tools – for adding comments explaining how your app works
- ▶ debug tools – to assist with identification and removal of program errors
- ▶ emulation tools – demonstrating your app running on a virtual mobile device
- ▶ testing tools – to show your app's performance, usage of resources etc.
- ▶ deployment tools – to transfer an app to its electronic store or physical mobile device.



▶ **Figure 7.6:** Android™ Studio is the official IDE for Android™ application development



PAUSE POINT

Can you explain what the learning aim was about? What elements did you find easiest?

Hint

Close the book and make a list of the different types and categories of mobile app available.

Extend

Which factors affect the design and features of a mobile app?

Assessment practice 7.1

A.P1

A.P2

A.M1

A.D1

Following in your footsteps, another junior developer is joining your development team next week. Although they have some programming experience, they have very little experience of developing mobile applications but are expected to learn very quickly.

You have been asked by your line manager to prepare and deliver an induction presentation that explains how the design of a mobile application is affected by its intended purpose and the needs, preferences and characteristics of the target users. In addition, you have been asked to provide an overview of current mobile technologies, explaining how these affect the design and implementation of such apps.

You should conclude your presentation with a worked example that analyses the design and implementation of some sample mobile apps, given these influencing factors, and which evaluates how they might have an impact on the effectiveness of the design and implementation.

Plan

- What is the task? What am I being asked to do?
- How confident do I feel in my own abilities to complete this task?
- Are there any areas I think I may struggle with?

Do

- I know what I am doing and what I want to achieve.
- I can identify when I have gone wrong and adjust my thinking/approach to get myself back on course.

Review

- I can explain what the task was and how I approached the task.
- I can explain how I would approach the hard elements differently next time. (ie what I would do differently).

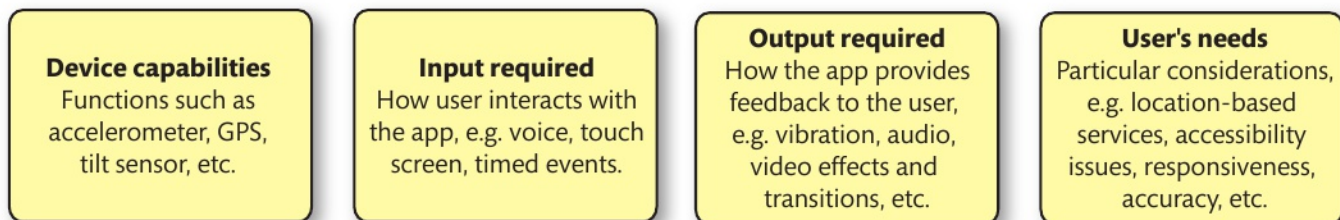
B Design a mobile app that utilises device functions

It is best practice to design your mobile application thoroughly before practical development takes place. In this section, we examine the design considerations and actions that you should make when designing your mobile application.

Analyse requirements for an app

Your first step when designing a mobile app is to consider the underlying computing requirements. Some of these are concerned with the capabilities of the device, while others focus on what the app needs to do (its core functionality) and the specific needs of the user. We can break this down using a quad diagram, which is a simple visual tool that focuses on four simple areas (see Figure 7.7).

It should be possible to complete a quad diagram for any prospective mobile app.



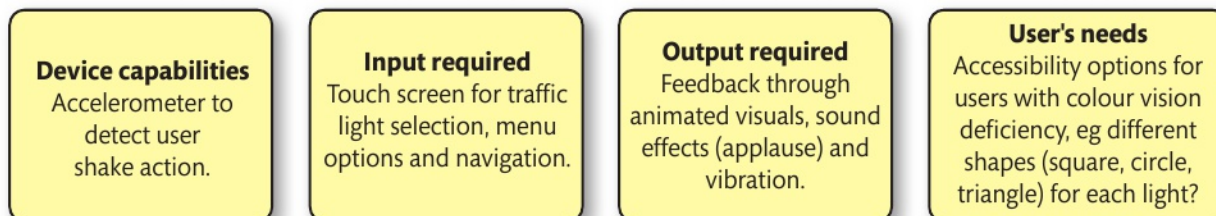
► **Figure 7.7:** Quad diagram for app requirements

Worked example: Traffic Lights

A simple mobile app is needed which will teach the correct colour sequence of a UK traffic light (a mandatory aspect of the UK driving test). Users should be able to see a timed animation of the UK traffic light sequence and be able to play a simple game where they predict which light will be next in the sequence by touching the correct light. Correct responses should be greeted with a congratulatory round of applause; incorrect responses should make the device vibrate. Accessibility options should be included for users with colour vision deficiency. The traffic lights are reset to the main menu via a simple shake gesture.

Step 1: Analyse and identify elements for each requirement category.

Step 2: Complete a quad diagram to organise your thinking.



Step 3: Check that all elements have been correctly identified and categorised.

Designing a mobile app

Creating appropriate documentation is an important and necessary part of the mobile app design process. If you are developing individually, it helps organise your thoughts and planning. If working as part of a team, it helps to communicate ideas, share problem solving and identify potential problems as they emerge.

Design documentation should minimally contain:

- ▶ actual user requirements
- ▶ a proposed solution.

User requirements

User requirements define the problem you are trying to solve. In fact, no attempt at problem solving should be made before you fully understand what it is that the user actually wants. Sometimes, it is necessary to narrow down user requirements based on available resources or simply confirm your understanding of them more comprehensively, usually by asking additional questions of the client or through market research of potential users.

Proposed solution

The proposed solution represents a blueprint of how the mobile app is going to be built. The design documentation should include all the details of the blueprint. In order to create a comprehensive software blueprint, there are many elements that you must include. These elements are outlined in the sections below.

Description of program tasks

The description of program tasks is a list of the core functionality of the mobile app, generated from the actual user requirements.

The tasks performed need not be listed in a chronological order; this is often impossible to achieve as functionality on a mobile application may be accessed in different ways. However, it should be comprehensive, meet the user's requirements as fully as possible and be usable as a 'to do' checklist once formal development begins.

Target platform(s)

The platform should identify the:

- ▶ required mobile device(s)
- ▶ operating system(s) including the targeted version
- ▶ type of app (native, web or hybrid).

Mobile devices include various smartphones, tablets or wearable technology but you may also need to specify particular versions. For example, if your app needs a front and rear-facing camera to work properly, this may limit it to certain models. Some apps may be designed for particular screen sizes; if this is true, make sure that the design documentation makes this clear.

In addition, although your mobile app development will certainly target a specific operating system (e.g. Android™ rather than Apple® iOS), you must ensure that any key software feature that you have used is not operating system version specific (eg Android™ 6.1) unless it is unavoidable. Doing so will limit potential users or force them to upgrade their operating system, if this is possible; some devices will simply become too outdated.

Screen layouts and navigation

Visual design is the cornerstone of good mobile app development. Visual design includes the principles of good screen layout and intuitive user navigation.



▶ **Figure 7.8:** Android™ wireframe examples

Many designers use graphical mock-ups of devices, along with their screens and widgets to prototype an app and receive feedback before any lines of program code are actually written. There exist online design suites that permit developers to prototype their screen layouts and navigation using simple 'drag and drop' functionality. The industry term for this process is **wire-framing** (see Figure 7.8).

Key term

Wire-framing – an important step in the screen design process, helping the developer to plan layout and user navigation using paper-based or electronic models of the devices and their visual components.

Research

Visit a sample wire-framing tool to explore the rapid creation of visual prototypes for your mobile app. Balsamiq® is one such tool and has a free demonstration available online. To access this website go to: <https://balsamiq.com/>

Algorithms

An algorithm is simply a set of instructions that can be followed to solve a problem or perform a calculation. Apps may be made from many different algorithms, implemented in the program code using a combination of many different programming constructs, functions and procedures. Algorithms used in mobile apps may be represented using a number of different design tools such as:

- ▶ pseudocode – an informal English-like outline of a program which can be converted to the target programming language
- ▶ activity charts – also known as unified modelling language (UML) activity diagrams, these demonstrate user activity flow through a coordinated set of actions, for example using a standard notation to login, purchase an item or book an appointment
- ▶ flowcharts – a graphical representation of the program, showing its actions and logic through a set of standardised symbols.

These design tools are not specific to mobile app development. The skills that you build up using them can easily be transferred to other programming environments.

Link

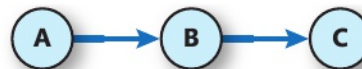
For more on different programming environments, see *Unit 6: Website Development* and *Unit 8: Computer Games Development*.

Control structures

The algorithms that control mobile apps are typically built using a combination of three basic programming building blocks or control structures.

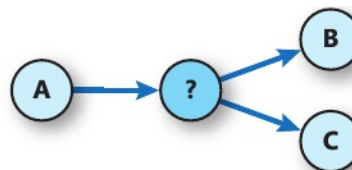
These control structures are sequence, selection and iteration.

- 1 Sequence – one action after another, none missed, none repeated (see Figure 7.9).



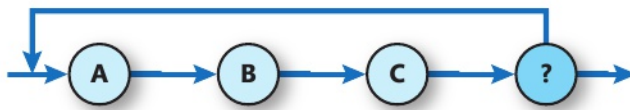
▶ **Figure 7.9:** A sequence

- 2 Selection – actions that are chosen based on a supplied condition which is evaluated as true or false (see Figure 7.10).

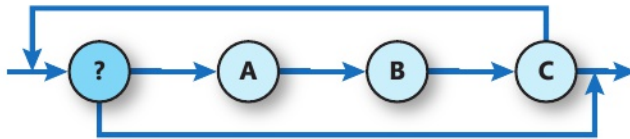


▶ **Figure 7.10:** A selection

- 3 Iteration – actions repeated a number of times, typically, until a condition is no longer true (see Figures 7.11 and 7.12).



► **Figure 7.11:** Post-conditioned iteration (or loop)



► **Figure 7.12:** Pre-conditioned iteration (or loop)

Data validation

Mobile apps typically use widgets such as on/off switches, pickers and list boxes for input, removing (as much as possible) the need for traditional keyboard input.

When text-based input does occur, it usually needs to be validated. Validation is simply checking to see if something makes sense before it is processed: for example, 'Age' must be entered using numeric digits only.

An inputted 'Age' of 21 is acceptable, but a value of F4 or typing 'twenty one' is not. Building validation rules, which check whether different inputs are sensible, into a program is very important as failing to do so can cause inaccurate results or, more severely, a **run-time error** or a fatal application crash.

Integration of device capabilities

If specific device capabilities are required (functions, interface aspects, operating system features or certain permissions) they should be documented as part of the design. It is important to know how, when and where they will be used in your app.

Alternative solutions

Few solutions exist without alternative approaches that also have some merit. A critical part of the design document should be coverage of different design solutions and delivery plans for the app. These may approach the solution from the opposite direction, target different devices or potentially require more or less resources (time, money, expertise) to complete.

Having alternative solutions should see you exhausting possibilities and, crucially, having a contingency plan in place should events not go according to plan during development of your chosen solution.

Resources and assets

Your design should detail any existing resources and assets that need to be incorporated into your app such as predefined code (yours and/or from a third-party library) and media assets. Your media assets may include:

- images (eg .jpg or .png image files)
- video (eg .avi, .mpg, .mov video clips)
- audio (eg .wav, .mp3, .au, .aiff audio files).

Care should be taken to ensure that media assets are suitably processed before inclusion. For example, they should be cropped to appropriate size and **optimised** for efficiency. Listing them as part of the design documentation also acts as your checklist for content preparation.

Key term

Run-time error – a problem that occurs while an app is being used, typically resulting in it locking (refusing to accept user input) or crashing (terminating and returning user to the device's menu or home screen).

Key term

Optimised – optimised assets are created using file formats which are more efficient as they require less storage space. This can result in improvements in performance and a smaller digital footprint on the device's resources. Examples include using .jpg images rather than .bmp files as these use data compression to reduce file size.

Test and review schedule

Scheduling robust testing and reviews is a critical part of the design process. As we will see, the planning of thorough test plans and the selection of suitable test data is essential for ensuring that the app you have built performs reliably and as expected on the targeted device(s).

Review is best achieved through the analysis of selected user feedback. The aim of reviewing your mobile app design is to help improve and refine it before it is developed and then formally released. By reviewing the design, you should be able to iron out any defects or niggles before development.

Constraints

Constraints are limiting factors that are encountered on a personal or team level or imposed by your targeted platform.

For example, you may be constrained by time, development costs, the available technology, or simply by your own technical expertise or that of your team.

You may also discover that your design is constrained by the selected platform. The permissions, capabilities and limitations of the operating system or hardware may force you to make particular design decisions.

Legal and ethical considerations

Legal and ethical considerations must be taken into account as part of your design documentation, particularly those relevant in the United Kingdom. The Data Protection Act 1998 (DPA) in particular should determine the way in which your application handles personal data. Privacy is important to users, so you must think about how your application collects, deals with, controls and secures personal data, and if it is absolutely necessary to collect specific types of data in the first place.

Ethically, there are concerns over data that can be shared by companies who develop apps. The personal data that is collected could be used to influence insurance, credit, education or employment decisions. As noted previously, modern mobile devices generally rely on permissions to prevent the users' personal data from being used by the companies who own the apps without their consent but, ultimately, the developer's guiding ethics are the final protection.

In addition, pay attention to potential issues such as copyright infringement, especially with regard to the use of media assets such as images, video and audio files. Although media assets can be easily incorporated into your mobile app, they retain their original creator's copyright and remain their intellectual property (IP), not yours.

Research

Visit the Information Commissioner's Office (ICO) for further reading on recommended codes of practice for mobile app developers, including ensuring users' privacy. To access this website go to: ico.org.uk.

Research

Visit the UK's legislative website and learn about the Copyright, Designs and Patents Act 1988. To access this website go to: www.legislation.gov.uk.



PAUSE POINT

Can you explain what the learning aim was about? What elements did you find easiest?

Hint

How do you define the design requirements for an app?

Extend

Which elements should be included in the documentation of a proposed solution?

C Develop a mobile app that utilises device functions

Once the design of a mobile app has been reviewed against the user requirements and accepted by the client, it is possible to start the physical development process. There are a number of different phases to work through that will challenge you to develop a rewarding range of practical and technical skills in mobile app development.

Content preparation for mobile apps

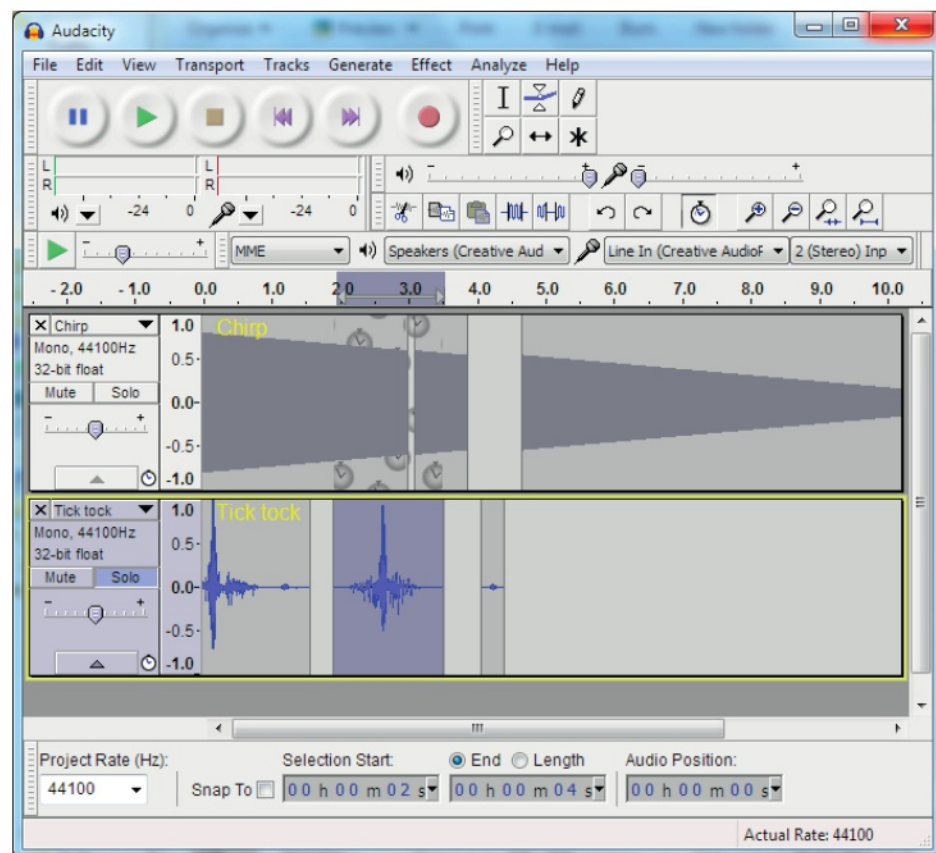
Mobile apps are assembled using assorted forms of digital content, typically including program code, visual layouts, sound files, images (icons, pictures, animation) and video.

Before you build your mobile app, it is a good idea to prepare this content, that is, your resources, so that you have them ready to put straight into your app.

Step by step: Content preparation for mobile apps

5 Steps

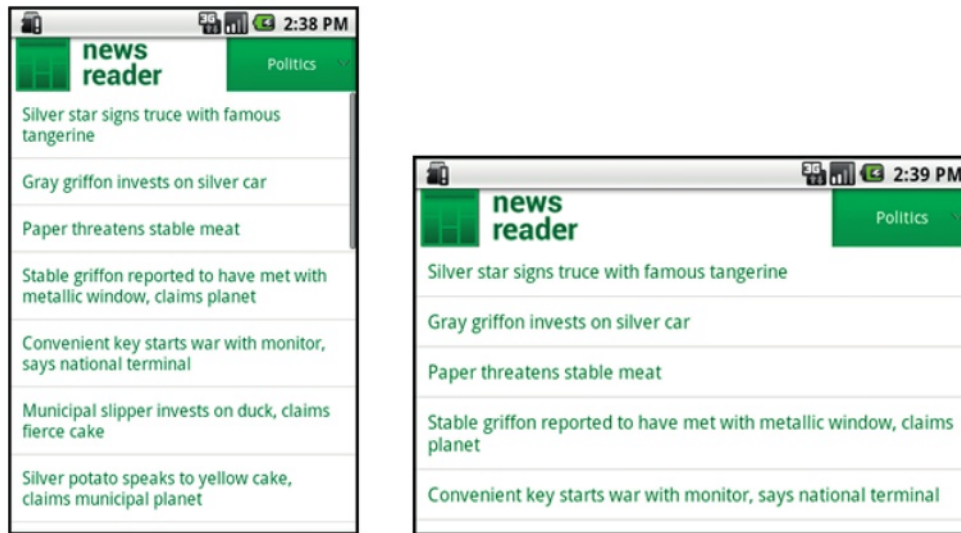
- 1 Select appropriate applications and techniques to prepare your resources (see Figure 7.13).



► **Figure 7.13:** Audacity® is a popular, open source, audio recorder and editing suite

Your code development options are likely to be limited to either Android™ Studio or Apple® Xcode. However, there are many different options for editing images and audio files, including online utilities, freeware and commercial software. Most editing processes have suggested workflows that encourage best practice in order to achieve the best results for the specific devices you are targeting with your app.

2 Consider how different device attributes will affect your content when it is used (see Figure 7.14).

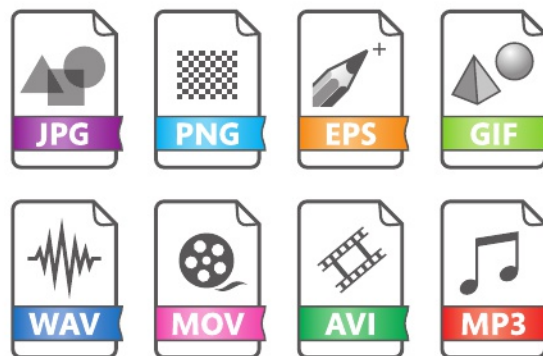


► **Figure 7.14:** Same app, different device sizes and orientation

Things to consider include:

- orientation (landscape or portrait or both)
- physical screen size (eg 7 inches)
- screen resolution (in pixels) (eg 1024 x 600)
- available app resources (e.g. RAM (eg 1 Gigabyte (GB))
- sound (eg whether a device has Dolby® support).

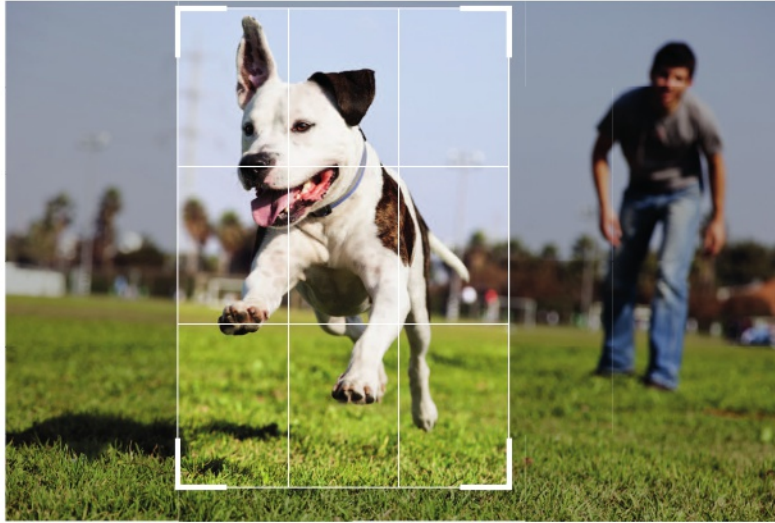
3 Choose compatible file formats for media assets (see Figure 7.15).



► **Figure 7.15:** Different file formats – not all may be supported by your target device

App development (and certain programming languages) may only support files in a particular file format (eg images may only be .jpg or .png). Be sure that your media assets are available in the correct format. If they are not, you can usually convert them using specialist software or using online tools. You should also consider the effects of user interaction on assets. For example, a user zooming-in on a raster image will experience the image appearing 'blocky' or overly pixelated, whereas vector images (if supported) do not suffer from this issue.

4 Optimise content as appropriate (see Figure 7.16).



► **Figure 7.16:** Optimising an image by cropping

Apply sensible optimisation to:

- program code (eg removing unnecessary sections of pre-written ('boilerplate') code that IDEs often provide as a helpful starting point)
- reducing file sizes of media assets by using compression techniques such as saving images as .png or .jpg files, selecting the most efficient format for a particular type of content or cropping parts of files that are not needed.

You should always remember that there is a trade-off between optimisation levels and quality (be too aggressive and the quality of your content will suffer).

5 Think about security.

Some data or assets may contain sensitive information. Consider using encryption tools to protect them and the target users of your app from cyber attack. Encryption is typically an arithmetic algorithm that scrambles sensitive data using a user-defined key. If the data is encrypted, it is considered to be safe from third-party prying because it cannot be correctly unencrypted without the user's key.

Reflect

Designing any mobile app involves substantial planning, particularly in terms of investigating and understanding the target users' requirements and the complexities of the target platforms. Any proposed solution should not just list the actions you need it to perform but should include an estimate of the amount of time each step requires for development.

Recording your findings accurately and comprehensively throughout the design and content preparation stages strengthens your problem solving by providing a solid foundation on which effective development may begin.

Tip

Always keep your user keys (passwords) safe and change them regularly. Even encrypted data could be at risk if passwords are not secure.

Link

For more on graphics (raster and vector images) see section Developing Computer Games (Worked Example) in *Unit 8: Computer Games Development*.

Developing a mobile app

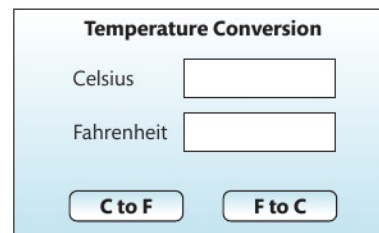
This section will take you through the process of developing a mobile app to meet identified user requirements and, as part of that journey, introduce you to the key concepts that form a mobile programming language and the development environment used to build it. This example uses Android™ Studio and the Java® programming language.

Reflect

Good communication skills are necessary for investigating and recording the target user's requirements. This may involve written and verbal communication, evidenced through activities such as conducting personal interviews, sending and responding to emails and the creation of formal design documentation for the mobile app.

Target users often know what they want in terms of a product but have very little technical awareness of the mobile app development process. You will have to reflect and learn how to adjust your verbal delivery to avoid the use of unnecessary jargon and find the language and technical level most suitable for the intended audience.

The following example will show you how to design a simple temperature conversion application that will allow the user to convert between readings in degrees Celsius and degrees Fahrenheit. The basic design can be seen in Figure 7.17.



The image shows a mockup of a mobile application interface for temperature conversion. It has a light blue background. At the top, the title 'Temperature Conversion' is centered. Below the title, there are two rows. The first row has the label 'Celsius' followed by a white rectangular input field. The second row has the label 'Fahrenheit' followed by another white rectangular input field. At the bottom of the interface, there are two rounded rectangular buttons. The left button is labeled 'C to F' and the right button is labeled 'F to C'.

► **Figure 7.17:** A simple design for a temperature conversion app

In order to create this app you will need to download and install the Java® SDK and a copy of Android™ Studio, both of which are freely available. The Java® SDK should be installed first, followed by Android™ Studio.

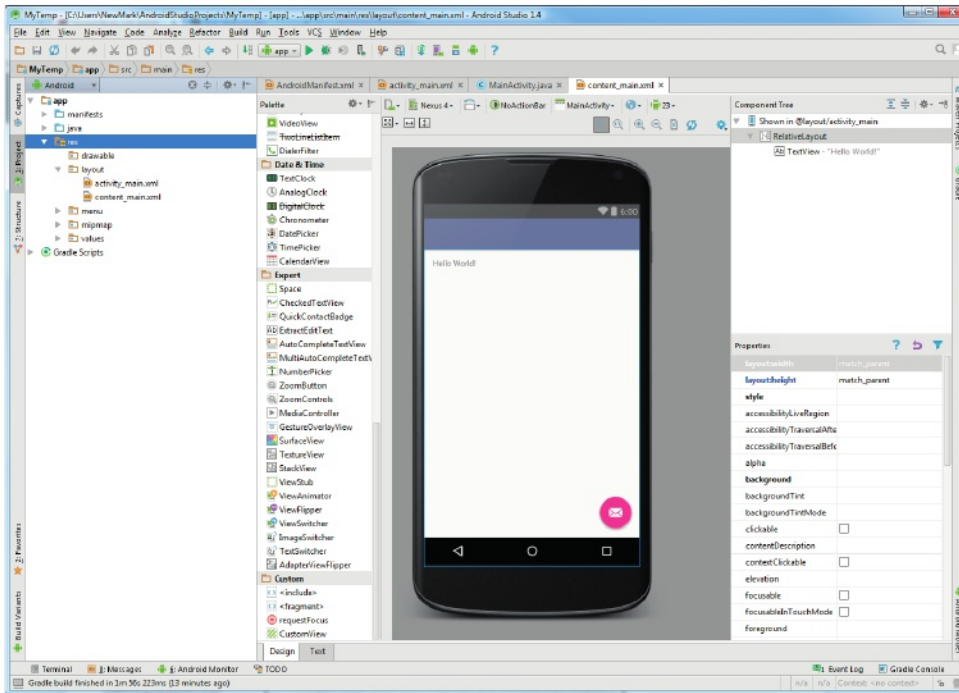
Link

Download Java® SDK from www.oracle.com

Download Android™ Studio from developer.android.com

Once you start Android™ Studio and create a new project, you are usually presented by a new project window (on the left). If not, select the Project vertical tab and then expand 'res' and then 'layout'.

Clicking on the content_main.xml entry should display your empty app with the default 'Hello World!' TextView in the central pane, as shown in Figure 7.18.



► **Figure 7.18:** Android™ Studio design view, showing the default 'Hello World!' TextView

Android™ Studio's design view works on a simple 'drag and drop' principle, so you should be able to move the selected TextView freely around on the app's main form.

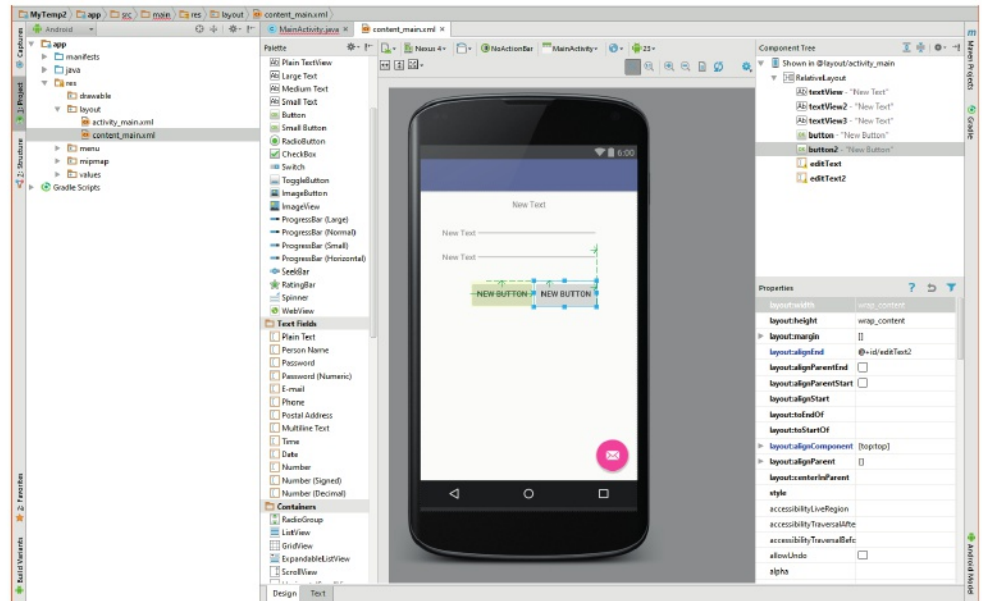
Android™ Studio has a number of different classes that may be used to place objects on the app's main form. These are shown in the Palette to the left of the central design pane. Classes are grouped into the following types.

- **Layouts** – control how the contents of a form are organised (eg in a table, in a grid, in rows).
- **Widgets** – different types of form element that are used to build the app's interface, allowing the user to input data, make selections and see output (eg TextViews, Buttons, CheckBox).
- **Text Fields** – specific types of TextView for defined jobs (eg Password, E-mail address, Telephone number).
- **Containers** – ways of grouping form elements together.
- **Date & Time** – different types of form elements related to the device's calendar and clock (eg DatePicker, TimePicker, CalendarView, TextClock, AnalogClock).
- **Expert** – complex types of form element for the more advanced app developer.
- **Custom** – a specialised class created by the developer or a third-party widget.

When you drag and drop a class from the Palette onto the form, Java® creates a solid object (a concrete instance of that class) which has a name, properties (things that describe it) and methods (things that it can do). Android™ Studio is also generating an XML (eXtensible Markup Language) file which describes the app's appearance as you make each change and addition; this is viewable on the Text tab at the bottom of the screen.

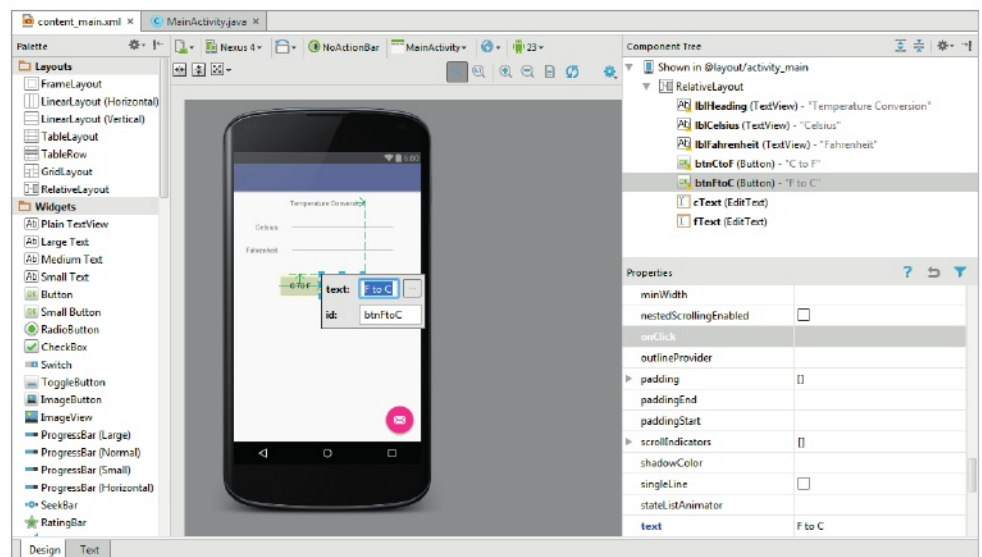
You will now delete the 'Hello World!' TextView and replace it with content based on your temperature conversion app design. This is shown in Figure 7.19.

- Edit the form's properties so that it resembles the design shown in Figure 7.19.



► **Figure 7.19:** The main form with additional Java® objects dragged from Android™ Studio's Palette

Each object can be edited by changing its properties (notice the Properties window in the bottom-right hand corner of the Android™ Studio IDE). Properties controls each object's visual appearance and allows you to alter their size, typeface and colour. It is also possible to double-click on a form object to edit its text and ID.



► **Figure 7.20:** Android™ Studio design view, showing form objects with new properties and IDs

You may have noticed that each object on your app's main form has a default name (or ID), for example textView, textView2, editText, editText2. Although this is helpful, it does not generate readable program code so it is a good idea to rename them sensibly. Do this by double-clicking on each object; a small dialog will pop up allowing you to change their IDs, as shown in Figure 7.20.

Make sure the new IDs match the list of objects, as shown in the Component Tree window in Figure 7.20.

Programming constructs

Now pause and focus on the Java® programming language in which Android™ apps are written.

Reserved words, local and global variables, constants and assignment

Java®, like most programming languages, has a number of reserved words. Reserved words are fundamental parts of the language which *cannot* be used by you for naming things. A list of common Java® reserved words is found in Table 7.4.

► **Table 7.4:** Alphabetical list of Java®'s reserved words (some are version specific)

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

When you name things, you are creating identifiers. There are two types of basic identifier – variables and constants. As the names suggest, variables store values which your app can manipulate. Constants store values that cannot change while the app is running.

Discussion

Mobile apps often use many variables and constants. Imagine creating a friendly app to calculate your wages for a part-time job. In a small group, discuss which values would be stored in variables and which would be constants, making sure that you justify your decisions.

In order to declare a variable we need to specify its name and its **data type**. Developers must select the most appropriate type for the data they wish to store. Table 7.5 shows Java®'s primitive data types and examples of different declarations.

Key term

Data type – the kind of data that we want to store in the mobile device's RAM. Storing each type of data requires different quantities of RAM. For example, the data type needed to store a number is different from the one used to store a character and will require different amounts of RAM. Exact names of data types can vary between different programming languages, so be careful to select the right one.

In each declaration we are saying:

data type variable name = value

The = sign (or assignment operator; see Table 7.5) is used to store the value in the named variable. The first time you do this is called an 'initialisation'; afterwards it is simply called 'assignment'. Variables may change their value many times as your app runs.

Link

For more on RAM see Memory and storage in *Unit 8: Computer Games Development*.

► **Table 7.5:** Java®'s primitive data types, their features and example declarations

Data type	What it is	Min – max range (where appropriate)	Default value	Example declaration
byte	8-bit signed two's complement integer	-128 to 127	0	byte age = 17;
short	16-bit signed two's complement integer	-32,768 to 32,767	0	short qty = 10000;
int	32-bit signed two's complement integer	- 2,147,483,648 to 2,147,483,647	0	int largeqty = 100000;
long	64-bit signed two's complement integer	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0L	long vlargeqty = 100000L;
float	single-precision 32-bit IEEE 754 floating point		0.0f	float wages = 2814.20f
double	double-precision 64-bit IEEE 754 floating point		0.0d	float salary = 28140.20d
boolean	Boolean 1-bit state; true or false	true or false	false	boolean alive = true;
char	single 16-bit Unicode character	'\u0000' to '\uffff'		char initial = 'A';

Most programming languages have the concept of local and global variables. The easiest way to think about this is to remember that global variables can be used anywhere in your app's code. Local variables are limited to being used in the block of code in which they are declared. However, Java® does things a little differently as you will see.

Key term

Operator – a special symbol (or multiple symbols) which tells the program to perform specific arithmetic, relational or logical operations on its data. Operators must be used in a specific order of precedence (this describes which is executed first). You may be familiar with this concept from using BIDMAS (or BODMAS) (Brackets, Indices/Orders, Divide, Multiply, Add and Subtract) in mathematics.

Operators

Operators are special symbols used to perform special tasks in most apps. If you are already familiar with symbols such as using * for multiply, then you are familiar with an arithmetic operator. Java® has a number of these and it is a good idea to familiarise yourself with them. The most commonly used operators in Java® are shown in Table 7.6.

► **Table 7.6 :** Common Java® operators, showing their grouping and purpose

Java® operators	Name	Operator group	Purpose
+	Add	Arithmetic	Adds two values giving their sum
-	Subtract	Arithmetic	Subtracts one value from another giving their difference
*	Multiply	Arithmetic	Multiplies two values giving their product
/	Divide	Arithmetic	Divides one value by another giving their quotient
++	Increment	Arithmetic	Increases a value by 1
--	Decrement	Arithmetic	Decreases a value by 1
==	Equal to	Relational	Tests where two values are equal
!=	Not equal to	Relational	Tests whether two values are unequal
>	Greater than	Relational	Tests whether one value is greater than another
<	Less than	Relational	Tests whether one value is less than another
>=	Greater than or equal to	Relational	Tests whether one value is greater than or equal to another

► **Table 7.6:** – *continued*

<=	Less than or equal to	Relational	Tests whether one value is less than or equal to another
=	Simple assignment	Assignment	Assigns a value to a variable, not to be confused with the double equal sign which tests equality
&&	And	Logical	Performs a logical And operation
	Or	Logical	Performs a logical Or operation
!	Not	Logical	Performs a logical Not operation

Control sequences

Recall the three control structures introduced earlier: sequence, selection and iteration.

When you are building a mobile app, sequences are represented by any block of Java® program code that executes line-after-line, with none missed or repeating.

Research

You can quickly experiment with the code samples shown by visiting a free, online Java® compiler. When you have keyed in the program code you want to try, simply click on the “Compile” and “Execute” options. You may even save your code for future use.

Visit www.tutorialspoint.com to try this out and discover new coding techniques, for example nested if statements and conditional operators.

Research

It is not possible to list all of Java®’s operators here. Visit the following online resource to find out about the other operators that exist and gather examples of their usage:

www.tutorialspoint.com/java/java_basic_operators.htm

Link

For more on sequence, selection and iteration see Control structures.

Selections are built using either an if...else or a switch statement. Which one you use depends on what you are trying to achieve.

A simple if...else statement has a condition that evaluates to either true or false. As shown in Figure 7.21, the actions in the first block are executed if the condition is true, otherwise the actions in the (optional) else block are used. The output from two test runs of this Java® code is shown in Figure 7.22.

```

1 import java.util.Scanner;
2
3 public class NumberTest {
4
5     public static void main(String []args) {
6
7         int number1;
8         int number2;
9
10        Scanner keyboard = new Scanner(System.in);
11        System.out.print("Enter 1st number: ");
12        number1 = keyboard.nextInt();
13        System.out.print("Enter 2nd number: ");
14        number2 = keyboard.nextInt();
15
16        if (number1 == number2) {
17            System.out.println("Numbers are the same!");
18        }
19        else {
20            System.out.println("Numbers are NOT the same!");
21        }
22    }
23 }
```

► **Figure 7.21:** Java®’s if...else statement

```

Enter 1st number: 4
Enter 2nd number: 5
Numbers are NOT the same!

Enter 1st number: 78
Enter 2nd number: 78
Numbers are the same!

```

► **Figure 7.22:** Output from two different tests of Java®'s if...else statement

Tip

Many modern programming languages are case sensitive and Java® is no exception. If you are having problems compiling your programs, check that you have used upper case and lower case letters correctly in your code. Missing out important symbols such as the semi-colons (which are used in Java® to separate statements) is also a common error, even for experienced developers.

Java®'s switch statement is a useful way to check for multiple values at the same time, as shown in Figure 7.23. It is also possible to add a default check in case no listed options are matched. Output from four test runs of the switch statement is shown in Figure 7.24.

```

1  import java.util.Scanner;
2
3  public class BankAccount {
4
5      public static void main(String []args) {
6          int option;
7
8          Scanner keyboard = new Scanner(System.in);
9          System.out.println("1 - Bank Balance, 2 - Account Query, 3 - Payments");
10         System.out.print("Enter option 1, 2 or 3: ");
11         option = keyboard.nextInt();
12         switch (option) {
13             case 1: System.out.println("You have chosen to see your balance");
14                     break;
15             case 2: System.out.println("You have chosen to make a query");
16                     break;
17             case 3: System.out.println("You have chosen to make a payment");
18                     break;
19             default: System.out.println("You have not chosen a correct option!");
20         }
21     }
22 }
23

```

► **Figure 7.23:** Java®'s switch statement

```

1 - Bank Balance, 2 - Account Query, 3 - Payments
Enter option 1, 2 or 3: 1
You have chosen to see your balance

1 - Bank Balance, 2 - Account Query, 3 - Payments
Enter option 1, 2 or 3: 2
You have chosen to make a query

1 - Bank Balance, 2 - Account Query, 3 - Payments
Enter option 1, 2 or 3: 3
You have chosen to make a payment

1 - Bank Balance, 2 - Account Query, 3 - Payments
Enter option 1, 2 or 3: 8
You have not chosen a correct option!

```

► **Figure 7.24:** Output from four different tests of Java®'s switch statement

Iterations (sometimes called loops) are represented by one of three common statements (for, while and do...while). Figures 7.25 to 7.27 show Java®'s for, while and do...while loops for a particular example. Figure 7.28 shows that the output from all three loops is identical.


```

1 public class ForLoop {
2
3     public static void main(String []args) {
4         int counter = 0;
5
6         for (counter = 1; counter <= 5; counter++) {
7             System.out.println("BTEC National");
8         }
9     }
10 }

```

► **Figure 7.25:** Java®'s for loop

```

1 public class WhileLoop {
2
3     public static void main(String []args) {
4         int counter = 1;
5
6         while (counter <= 5) {
7             System.out.println("BTEC National");
8             counter++;
9         }
10 }
11 }

```

Figure 7.26: Java®'s while loop

```

1 public class DowhileLoop {
2
3     public static void main(String []args) {
4         int counter = 1;
5
6         do {
7             System.out.println("BTEC National");
8             counter++;
9         } while (counter <= 5);
10 }
11 }

```

► **Figure 7.27:** Java®'s do...while loop

```

BTEC National
BTEC National
BTEC National
BTEC National
BTEC National

```

Figure 7.28: Output from all three loops is identical

Although all three loops generate the same output, they have all worked differently. Ideally, you would use for loops when you know that code needs to repeat a fixed number of times. While loops may not even run once (they are pre-conditioned) and do...while loops always repeat at least once (they are post-conditioned).

Loops repeat based on their controlling condition being true. In the example shown in Figures 7.25 to 7.28, the controlling condition is the counter variable having a value less than (or equal to) 5. When the counter reaches 6, the condition is no longer true and the loop ends.

Functions and procedures

Another key component of a mobile app is the concept of a function or procedure.

Functions and **procedures** are common features in procedural programming while, in **object-oriented** programming, they are essentially represented by the methods which occur with the class.

Key terms

Function or procedure – a function is a block of code, ideally between 5 and 50 lines in length, which has a single defined purpose. Although written just once, it can be executed many times during a program by using a function 'call', reducing the need for repeated code. In some programming languages, the terms 'function' and 'procedure' are used interchangeably, but, in others they are very different concepts: a function typically returns a calculated value while a procedure performs a single identifiable task.

Object-oriented - Java®, which we use to create Android™ apps, is categorised as a class-based, object-oriented programming (OOP) language. In OOP, objects are created from classes that are usually modelled on real-world 'things'. Each class acts as a software blueprint, encapsulating (or containing) the thing's state (its data or properties) and behaviour (its functions or methods) in program code.

Tip

The temperature conversion app is relatively straightforward and should just make use of a basic sequence control structure. When you start designing more complex apps, the other constructs (sequence and iteration) will become important, so remember the if, switch, for and while statements shown here.

Objects and classes

Classes are a fundamental aspect of every Android™ mobile app solution created using Java®. Classes are used to represent the physical components that you use to build an app. For example, each item in the Android™ Studio Palette (screen components such as forms, Layouts, Widgets, and Text Fields) represents a different class that you can use.

Once the class is dragged to the form, a concrete instance of that class is created, which we call an object. Try to think of a class as a design pattern, something like a jelly mould, stencil or cookie cutter; one jelly mould (the class) can be used to make many jellies (the objects).

Each object has associated methods and properties and can interact with other objects to perform a task. Commercial apps will use a wide range of classes, many of which may be freshly created by their developers.

Event handling

Once you have created the visual design for your app's interface and named objects sensibly, the next step is to code the algorithms that calculate the outputs or actions needed. In the case of our temperature conversion app, this means converting between a temperature input in degrees Fahrenheit to an output in degrees Celsius and vice versa.

These calculations are executed when a user presses one of the two buttons we have added to the app's main form. When the user presses a button (or performs any action the operating system recognises) it triggers a specific event – in this case, what is commonly called an 'on click' event.

In order to link the event trigger to the calculations, you need to build an event handler. There are many ways to achieve this in Android™ Studio. The most traditional method involves registering a listener method for each button that will run a specifically coded **event handler** method.

Revisit the Project explorer pane, expanding 'Java' and then 'MainActivity'. Clicking on this should display the Java® program code, which will have been automatically created for you. You will need to insert the highlighted code shown in Figure 7.29 to add appropriate event handlers that will respond to user inputs and calculate the desired outputs, in order to get the right screen components.

Key term

Event handling – describes the process of using a specially written function or class method to perform set actions when a user or system event is triggered. For example, outputting the result of a calculation when a button is pressed or displaying a device's 'low battery' warning indicator when it falls below a certain charge level.

```
import android.support.design.widget.Snackbar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
                    .setAction("Action", null).show();
            }
        });

        addButtonCtoFListener(); //call method to add listener to CtoF button
        addButtonFtoCListener(); //call method to add listener to FtoC button
    }
}
```

► **Figure 7.29:** Adding new code to your app

The new (highlighted) lines shown in Figure 7.29 perform two actions.

- 1 They import various Java® code packages that we need for our app to compile into our program.
- 2 They call two listener methods, one for each button on the app's form.

Your next task is to create the Java® code for the two event listener methods. Each will include an event handler that:

- ▶ gets the value entered into a TextField on the app's form
- ▶ performs the correct calculation
- ▶ puts the results back into the other TextField on the app's form.

Figures 7.30 and 7.31 show the new code for each listener and handler. You should note that the Java® code for the `addButtonCtoFListener` method is annotated to demonstrate best practice.

```

/*
 * Class method adding a Listener to
 * the CtoF button to enable responding to
 * events
 */
public void addButtonCtoFListener() {

    //Create button object representing CtoF button on form
    Button btnCtoF = (Button)findViewById(R.id.btnCtoF);

    /*
     * OnClick listener for CtoF button will
     * call its associated event handler to perform
     * the CtoF calculation
     */
    btnCtoF.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            float celsius;    // Inputted Celsius value
            float fahrenheit; // Calculated Fahrenheit value

            //alias Celsius input box
            EditText cValue = (EditText) findViewById(R.id.cText);

            //grab celsius value from input box, converting from text to floating point
            celsius = Float.valueOf(cValue.getText().toString());

            //perform conversion of Celsius to Fahrenheit
            fahrenheit = (celsius * 9.0f / 5.0f) + 32.0f;

            //alias Fahrenheit input box
            EditText fValue = (EditText) findViewById(R.id.fText);

            //convert calculated Fahrenheit floating point value to text
            String fahrenheitText = Float.toString(fahrenheit);

            //put the converted Fahrenheit text value into the Fahrenheit input box
            fValue.setText(fahrenheitText);
        }
    });
}

```

▶ **Figure 7.30:** Java® code for the `addButtonCtoFListener` method (annotated)

```

public void addButtonFtoCListener() {

    Button btnFtoC = (Button)findViewById(R.id.btnFtoC);

    btnFtoC.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            float celsius;
            float fahrenheit;
            EditText fValue = (EditText) findViewById(R.id.fText);
            fahrenheit = Float.valueOf(fValue.getText().toString());
            celsius = (fahrenheit - 32.0f) * 5.0f / 9.0f;
            EditText cValue = (EditText) findViewById(R.id.cText);
            String celsiusText = Float.toString(celsius);
            cValue.setText(celsiusText);
        }
    });
}

```

▶ **Figure 7.31:** Java® code for the `addButtonFtoCListener` method (not annotated)

Reflect

Read through the annotated code for the `addButtonCtoFListener` method and compare it with the similar, but uncommented, code in `addButtonFtoCListener`. Could you add appropriate comments to the `addButtonFtoCListener` method to improve its readability?

Code annotation

As you have seen, it is important to ensure that your code is suitably documented. Annotating code essentially means inserting developer-readable comments throughout your code. Although these comments are removed during the compilation process and therefore inaccessible to the user, their inclusion is good practice.

The purpose of annotating your program code with meaningful comments is to improve its readability and aid understanding. This can be especially important when programmers are asked to redevelop apps that were originally written by others.

Your comments should make clear how your code relates to its real-world application, not explain the syntax of the actual reserved words used. The actual techniques used to comment on program code vary between program languages.

Java® uses a combination of multiline and single line comments, and this is shown in Figure 7.32. Another popular use of annotation is to 'comment out' program code that the developer wants to keep but not compile, possibly as they test new ideas or identify code which is not working correctly.

```
1 public class HelloWorld{
2
3     /*
4      *   Main function
5      *   Program to test program output
6      *   Written by A. Jones
7      *   Version 1.0
8      */
9
10    public static void main(String []args){
11
12        // Output welcome message
13        System.out.println("Hello World");
14    }
15 }
16
17
18 }
```

► **Figure 7.32:** Multiline and single line comments

The use of meaningful identifiers is also a key aspect of code annotation as the representation names you have chosen are said to self-document.

Some SDKs and third-party tools can generate web page documentation automatically from your program code if it is formatted in the correct fashion. For example, Oracle®'s Javadoc tool performs this function for Java®.

Research

Visit www.oracle.com to investigate Oracle®'s Javadoc tool and learn how to incorporate Javadoc-compatible comments into your code and automatically generate navigable web pages for them.

Utilise device capabilities

Android™ and Apple® support mobile app developers by providing **application programming interfaces (APIs)** and **frameworks** that permit apps to receive motion data from integrated device hardware such as gyroscopes and accelerometers. Android™ calls this the Sensors framework while Apple® refers to their implementation as the Core Motion framework.

Key terms

Application programming interface (API) – this acts as a library of pre-written routines that provide the developer with access to other systems (such as databases), the operating system of a device or its actual hardware. The aim of an API is to make the development process easier by abstracting (hiding) the complexity of the software and hardware systems beneath.

Framework – a particular set of development tools that can be accessed via its API.

Interrogate device status

A key part of mobile app programming is the ability to interrogate the device's status. Different types of status may be checked by the developer, including important statistics such as its physical location, battery level or orientation (that is, whether it is portrait or landscape). For example, Android™ uses a useful BatteryManager class, which provides a method for querying battery and charging properties.

Research

Many examples can be found that provide the Java® code necessary to interrogate an Android™ device's status. Visit the www.tutorialforandroid.com website that demonstrates how the battery level of such a device may be checked.

Investigate this code, adapt it and incorporate into your own projects.

Although you do not need to perform this type of interrogation in your introductory temperature conversion app, being able to do so is an important skill.

Orientation of device

Your app needs to auto-detect the orientation of the physical device on which it is running (and when it changes). It does this by querying the screen's rotation. The device's normal rotation is derived from its screen format: for example, if the screen is naturally tall (portrait) and the device is turned sideways (landscape) its rotation value will either be 90 or 270 degrees depending on the direction it was turned (clockwise or anticlockwise). Helpfully, Android™ supplies an appropriate class called 'Display' within its API to assist developers determine the screen rotation, size and refresh rate.

Apps can also force the orientation mode by automatically changing the device's rotation to fit a portrait or landscape view, although this is not generally considered to be good practice.

Research

Investigate Android™'s 'Display' class and its methods by visiting developer.android.com

Creating an executable for a target device

You can deploy your app to either an emulated device or the actual physical device you have targeted.

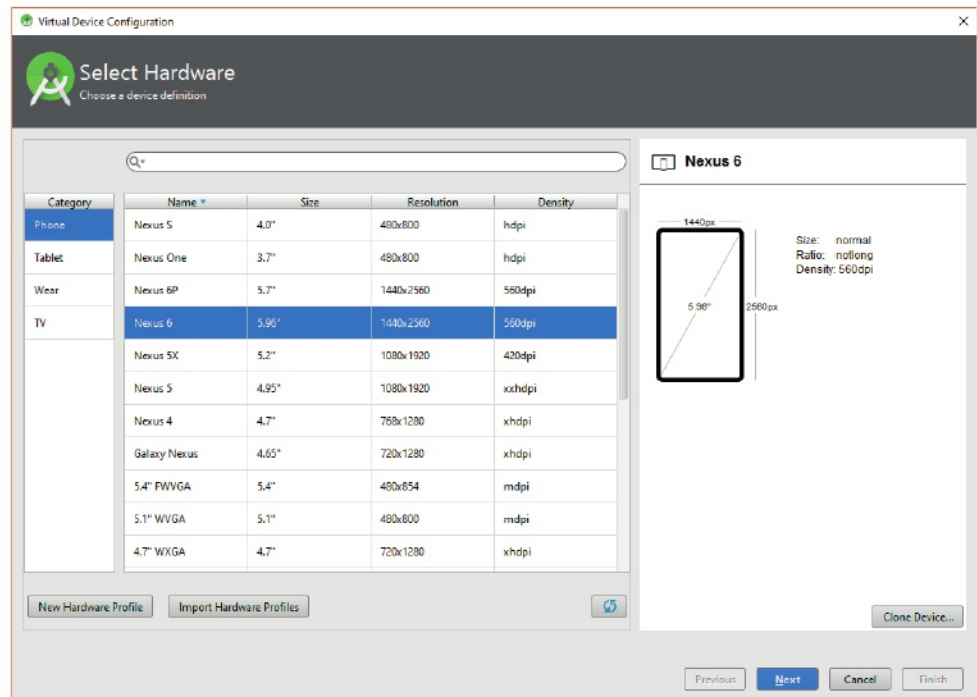
Emulated device

Using an emulated device is useful because it lets you test the compatibility and performance of your app on types of device to which you may not have physical access, for example different smartphones, tablets, TVs and pieces of wearable technology. In Android™ Studio, you can achieve this by creating an **Android™ virtual device (AVD)**. To create an AVD it is necessary to specify the hardware, and the version of the Android™ operating system being used (4.X KitKat, 5.X Lollipop or 6.0 Marshmallow).

Android™ Studio allows you to create multiple AVDs and then choose which one you want to use when developing your app. This allows you to test your app easily on many different physical devices. The device can be chosen through the available hardware definitions, as shown in Figure 7.33.

Key term

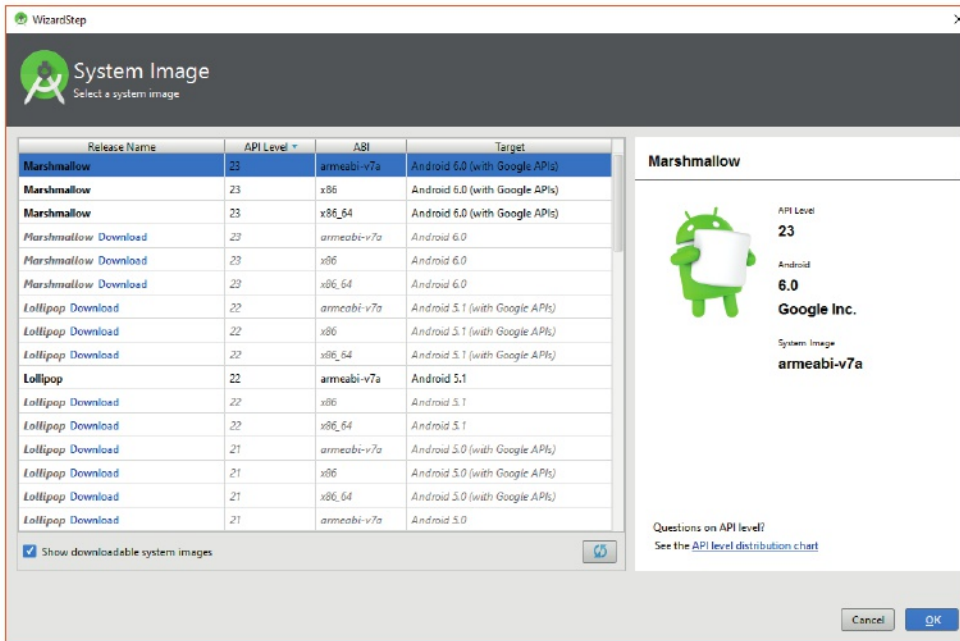
Android™ virtual device (AVD) – a software emulated version of a particular physical device, running within a host operating system such as Microsoft® Windows® or Apple® OS X®.



► **Figure 7.33:** Selecting a hardware definition for a real Android™ device

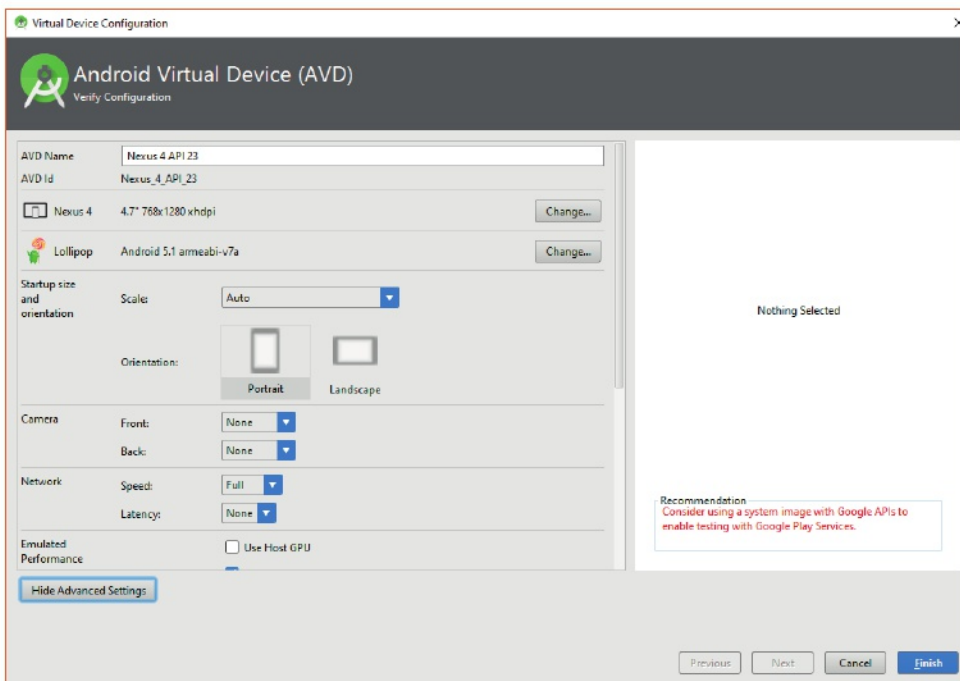
It is then possible to select an Android™ system image to use. Android™ Studio will have some images already installed but you may download others as you need them (as shown in Figure 7.34). In order to run your app successfully using an AVD, you must use a system image that is compatible with the programming or hardware features you have used.

Each system image is typically available in different virtualisation options: x86 (32-bit), x64 (64-bit) or ARM (Advanced Reduced Instruction Set Computing Machine). ARM is generally the slowest but is more likely to be independent of virtualisation options set in your host PC's BIOS (Basic Input Output System) (these should always be modified with caution).



► **Figure 7.34:** Selecting an Android™ system image for the AVD

It is then possible to tailor various options to your personal preferences, as shown in Figure 7.35.



► **Figure 7.35:** Setting preferences for your AVD

Figure 7.36 shows the AVD running in Microsoft® Windows® 10.

Physical device

The other technique for testing your Android™ app is to connect the system upon which you are developing it (for example Apple® Mac® or Microsoft® Windows® PC) to a



► **Figure 7.36:** An AVD running in Microsoft® Windows® 10



► **Figure 7.37:** Android™ device connected via USB cable

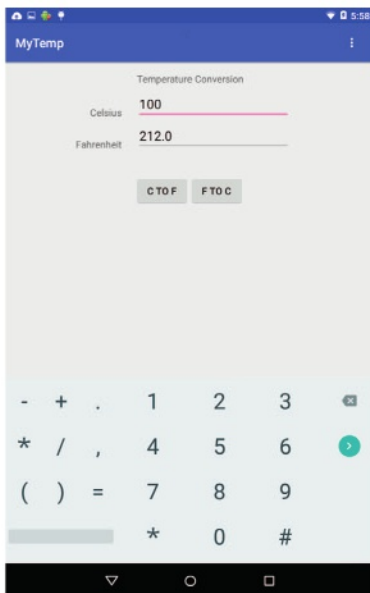
physical Android™ smartphone, tablet, or wearable technology via a compatible USB (Universal Serial Bus) cable, as shown in Figure 7.37.

In addition to installing the correct Android Debug Bridge (ADB) driver for your host system to talk to the Android™ device, it is often necessary to modify some of the device's settings in order to run your app. This often involves enabling options for USB or running apps not downloaded through Google Play™, and only has to be performed once. These developer settings can vary from device to device, and may even be deliberately hidden to prevent accidental damage, so you are advised to check the correct procedure with the device's manufacturer before attempting this.

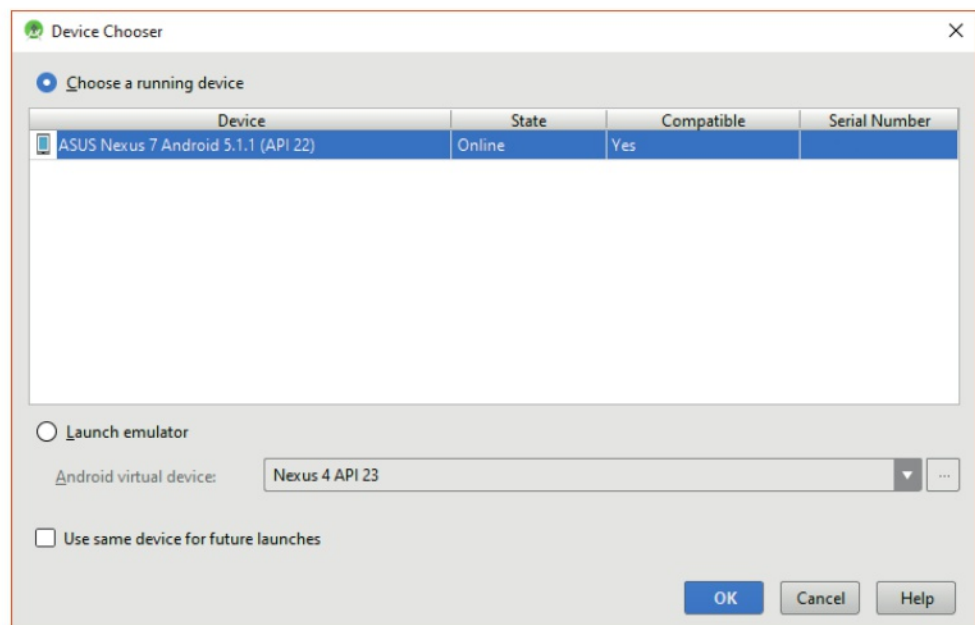
The primary advantage of running your app on the target physical device is that you will get the truest impression of its performance and behaviour. Many devices also have developer debug tools that can assist your live testing.

As a developer, you just need to select the connected Android™ device (not an AVD) once you have clicked Android™ Studio's green 'run' app button. If the device is successfully connected, it should be listed as a running device, which can then be chosen. Once you have selected it and clicked the OK button, Android™ Studio will then start to transfer the necessary Java® code to the device via USB and start running the app; the process is relatively quick.

Figure 7.38 shows that a physical device has been selected as the target platform and Figure 7.39 shows the temperature conversion app transferred and running on a physical device.



► **Figure 7.39:** Your app transferred and running on a physical device



► **Figure 7.38:** Selecting a physical device as the target platform

The physical device's virtual keyboard automatically appears as the app runs, limiting the user to entering only numbers. This happens because of the design decision to choose 'Number (Signed)' text fields.

Although a quick test seems to indicate that the app is working correctly (100 degrees Celsius is indeed equivalent to 212 degrees Fahrenheit), your next step should be to review your efforts through a process called quality control.

Quality control

Quality control is a process that reviews the standards of manufacturing applied during production. Its aim is to deliver the best product or service for a customer. For

a mobile app, this essentially means measuring how well the design was implemented during development for the target device(s).

Quality control is an intrinsic part of the mobile app development process, helping you to assess the resulting standard that your development process has achieved and, crucially, to identify areas of potential improvement. Quality control should be performed before formal testing starts. An overview of quality control is shown in Table 7.7.

► **Table 7.7:** Quality control areas of concern, key questions and considerations

Area	Question	Consider...
Efficiency and performance (This is called application 'profiling')	How does your app impact on the resources of the mobile device? Can we identify any issues affecting performance?	<ul style="list-style-type: none"> • use of RAM allocation • use of device storage media • use of central processing unit (CPU), particularly in terms of its percentage share • use of system and application processes that may highlight performance bottlenecks • use of graphical processing unit (GPU), especially when rendering user interface (UI) elements • use of battery, comparing drain of similar applications • temperature constraints – mobile devices can shutdown to protect components during periods of heavy use (especially when recharging) if they become too hot.
Maintainability	How easy is it to modify or improve your app? Have you programmed the app appropriately?	<ul style="list-style-type: none"> • recommended standards of coding • levels and usefulness of code annotation • structure and organisation of your program code • extensibility of your solution • use of programming constructs, functions, procedures, data types, classes and the device's code framework.
Portability	Which devices are compatible with your app? How could you improve the potential compatibility?	<ul style="list-style-type: none"> • devices which will correctly run your app • devices which do not run your app (and identify the reason(s) why) • devices which run your app but have minor issues (and what can be adjusted to improve portability) • any unexpected compatibility issues which you had not anticipated • whether your app's actual portability meets the platforms targeted in the design phase (including specific versions of the operating system).
Usability	How easy is it for a user to interact with your application? How can you improve the user experience?	<ul style="list-style-type: none"> • adherence to manufacturer guidelines, ie Apple® iOS and Android™ recommended design principles and standards • capturing user feedback, both good and bad • identifying common likes and dislikes • requesting user feedback for improvement • identifying whether the app meets the user requirements targeted in the design phase.

Links

- For more on RAM, CPU and GPU see the section on Hardware in *Unit 8: Computer Games Development*.
- For a reminder of what is meant by 'user interface (UI)', see *User interface*.

Quality control is best achieved through manual (rather than automated) review and can be a self-reflective process. Formal testing will examine feedback from a wider range of users.

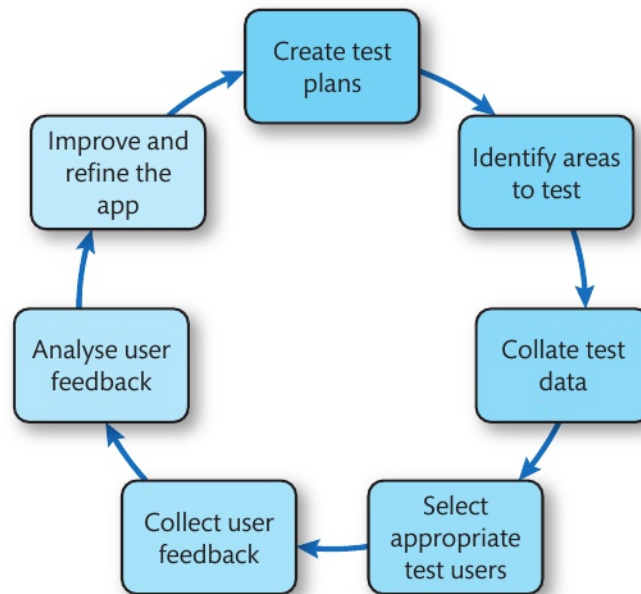
Reflect

The quality control process should help you to assess the standard of your app, both in terms of its design and its implementation. In doing so, it should provide a vital insight into your own performance levels, by highlighting the target areas that have been met and those which need some improvement.

Reflective quality control is a key component of any self-improvement process and is a requirement for achieving higher grades and, eventually, of improved performance in the workplace. Reflect upon the design and implementation of the development of the Temperature Conversion App. How could it be improved?

Testing a mobile app

Testing is widely recognised as an essential process that ensures that any mobile app you develop meets the requirements that were identified during its design, and that it operates in an accurate, reliable and robust manner. Testing should be performed every time the app is updated, so the process should be seen as cyclical; this is shown in Figure 7.40.



► **Figure 7.40:** The cyclical stages involved in robust testing of a mobile app

Test plans should detail in a step-by-step fashion how you are planning to test your mobile app. This will include the identification of several 'use cases'. Each use case attempts to tell the story of a user's interaction with your app (successful and unsuccessful) and typically includes a combination of:

- test data being entered
- operations being performed
- the order of the operations being performed
- the user's responses to your app's prompts.

Test data typically can be found in one of three possible states.

- Normal – data within the acceptable range that the user would normally enter.
- Extreme – unlikely data at the edges of the acceptable range.
- Erroneous – data which should not be entered (eg text rather than numbers).

Another concept that you may encounter as part of the test plan is the idea of white and black box testing. White box testing is usually performed by the developer by tracing the use cases through the program code logic and completing **trace tables**. A black box test is performed by a user by following a use case. The user has no exposure to the program code and does not need to know how the app works, whereas you, as the developer, are only interested in the outcome that the user gets.

Areas which need to be tested include the following.

- ▶ **Functionality** – all functions in the app should work as expected.
- ▶ **Acceptance** – the app should be fit for purpose: that is, it should meet the identified user requirements.
- ▶ **Installation** – covers the app's initial installation (typically from an app store download), its updating and its eventual removal.
- ▶ **Performance** – how well the app behaves and impacts upon the target device(s).
- ▶ **Usability** – the app should be intuitive to use and its interface quickly mastered by its users; getting a positive response from testers is often a key part of an app's success.
- ▶ **Compatibility** – the app should work consistently well across different models and brands of device.

Test users and user feedback

It is advisable that you select test users who have had very little to do with the development process. This will ensure that their observations are honestly expressed and without the bias that project involvement typically brings. Users should be drawn from a random population sample, ideally with mixed levels of technical experience.

User feedback must be collected from the user. This may be achieved using a variety of fact-finding techniques including:

- ▶ direct observation
- ▶ one-to-one interviews
- ▶ focus groups
- ▶ questionnaires or surveys
- ▶ automated error reports generated by the app itself.

Collecting user feedback can be both time-consuming and expensive, so techniques have to be used wisely. A common (and cost-effective) technique is the logging of feedback from confirmed users using online app stores because it is free and unsolicited (for example from Apple®'s App Store or Google Play™). The release of free Alpha or Beta (very early) releases of an app to selected users is also a useful tactic to consider.

Feedback should, ideally, be a mixture of quantitative and qualitative data which can be collated and analysed to produce useful information and identify patterns and trends in the user experience, which can highlight faults and areas for improvement.

Quantitative data – data that specifies quantities: that is, it can be measured and written as numbers, percentages or proportions. For example, '50 per cent of users experienced no issues while using the app'.

Qualitative data – are represented by reasons, opinions and motivations and can be used to explore (and make sense of) quantitative data. For example, 'users found the application difficult to navigate and text hard to read on smaller devices'.

Key term

Trace table – a table created by the developer, through white box testing, which charts the changes in program variables as a use case is performed. The trace table would have entries for each use case with expected (what should happen) and actual (what does happen) outcomes. Comparing these outcomes helps the developer assess whether the app is working correctly.

Discussion

Look at the two examples of quantitative and qualitative results. Can a link be found? Is the reason why 50 per cent of users experienced issues when using the app (quantitative result) explained by the problems encountered with the navigation and text size (qualitative result)? Discuss in small groups.

Table 7.8 Shows quantitative and qualitative questions in a questionnaire.

► **Table 7.8:** Quantitative and qualitative questions

Question 10a	How was MyApp's user interface?	<input type="checkbox"/> poor <input type="checkbox"/> acceptable <input type="checkbox"/> good <input type="checkbox"/> outstanding
Question 10b	Please explain your reasons ...	

For example if users' feedback suggested that while some found the user interface 'attractive and with a bright colour scheme' (qualitative data), you also find that '60 per cent of users judged the user interface 'poor' (quantitative data), then there is clearly a usability issue in your app's design.

If the supporting data is not available, it might be necessary to return to those users and investigate their unsatisfactory responses to find the underlying cause. Superior fact-finding should always ask for reasons and not just record a score.

Improving and refining your app

The process of improving and refining your app is guided by analysing the combined results of your testing (that is, done by the developer) and the received user feedback.

Issues to tackle should include:

- ensuring that the app has all the agreed functionality described in the design
- fixing run-time errors, bugs or application crashes to improve reliability
- fixing inaccurate calculations and outputs
- improving responsiveness
- improving user interface (eg colour scheme, typeface, font sizes, images)
- improving user navigation and app flow
- increasing speed/performance
- reducing the app's use of device resources by further optimising assets or simply programming more efficiently
- improving the compatibility of the app with different models/brands of mobile device.

Reflect

You will have reviewed your app using feedback from fellow professionals and test users. The views expressed by others are often very useful as they are typically unbiased, providing a neutral judgement about the design and implementation of your app. Learn to respond to outcomes and take this feedback in your stride. This will enable you to react in a mature fashion that leads to improvements in both the end product and your problem solving abilities.

Any changes made to the app should be recorded in a change log, which is a document written by the developer that details the fixes and improvements made for each version of the app. A version number is used to clearly identify each successive release. This uses a 'major.minor' system, for example:

- MyApp ver 1.0 – first major release
- MyApp ver 1.1 – minor update, fixing menu (from user feedback), improving compatibility
- MyApp ver 2.0 – major update, includes additional functionality.

As you can see, minor versions increment when small updates and fixes are committed (for example MyApp ver 1.1). Many minor releases may occur between major updates. In contrast, major updates represent large changes to the app functionality, features or code (for example, MyApp ver 2.0). You may also have personal experience of later app

releases introducing brand new errors and bugs. This is quite common and a hazard that every developer faces when trying to remedy an identified problem; in fixing a bug, they sometimes introduce others.

Lessons learned from developing a mobile app

It is important that you reflect and evaluate the effectiveness of the app that you have developed. This review encourages you to ask yourself the following questions.

- ▶ How well has the solution met the user requirements identified in the design document?
Think about: how has your app met the targeted user requirements, which elements has it achieved and which elements are missing or incomplete?
- ▶ Which issues arose during testing and your refinement of the app?
Think about: the issues you should have anticipated, how you could have prevented these problems from occurring, which coding techniques or device features could have been used better?
- ▶ How could the app be improved to meet the user requirements more closely?
Think about: the testing and user feedback, the aspects that could be added or refined.
- ▶ With hindsight gained through your testing and user feedback, if you were to repeat the task, were there alternative solutions that should have been implemented instead?
Think about: the testing, user feedback and the alternative designs in your design document.

Whatever the outcome, there is always room for improvement when developing a mobile app. One of the key parts of the learning process (and of developing better apps) is learning from your mistakes.

Reflect

Mobile app developers need to develop and demonstrate behaviours that fit the professional nature and expectations of the industry. Some key attributes you must adopt are:

- giving respect to others, especially their opinions when they are giving you essential feedback
- communicating appropriately with a target audience, providing and accepting open and honest ideas and opinions
- having an open mind to new ideas and alternative solutions and not being afraid to try new techniques to improve a solution or the target user's experience
- a commitment to personal improvement, by developing both new knowledge and practical skills
- the desire to achieve excellence through the standards of problem solving applied and the mobile apps created.



PAUSE POINT

Can you explain what the learning aim was about? What elements did you find easiest?

Hint

Describe the general steps of developing an app from a design.

Extend

What is the difference between quality control and formal testing?

Assessment practice 7.2

B.P3

B.P4

C.P5

C.P6

C.P7

B.M2

BC.D2

C.M3

BC.D3

You have been asked by your line manager to lead a development project for the design and implementation of a new mobile app that generates quotations for a landscape gardener. This involves selecting the prices of services that are involved (eg gardening, paving, planting, mowing) and the costs of goods (eg seeds, plants, bushes) that are to be included. Prices for the various services should be selected from local businesses although they should be easy to configure in the app itself. A standard rate of VAT is also expected.

Intending to demonstrate best practice to a newly recruited junior developer, you start by producing designs for a mobile app that meet the requirements you have identified. As part of your design process, you should review the design with your peers to identify opportunities for improvements or refinements. Before commencing development, you should justify and evaluate your design decisions to your junior colleague by explaining how they meet the identified requirements.

Once the design has been agreed, you must produce the mobile app from your design, testing it for functionality, usability, stability and performance. The working app should then be reviewed to ensure that it meets identified requirements and is optimised as necessary.

A final evaluation should be made of the optimised mobile app, comparing and contrasting it to the client requirements.

Plan

- What is the problem I am being asked to solve? Do I understand the client's needs?
- How confident do I feel in my own abilities to complete this task, in terms of design and coding?
- Are there any technical areas I think I may struggle with? If so, do I know how to research these?

Do

- I know what it is I am doing and what I want to achieve.
- I can identify through feedback and review when I have gone wrong and adjust my thinking/approach to get myself back on course.
- I am prepared to optimise my solution based on identified requirements and feedback.

Review

- I can explain what the task was and how I approached the task, justifying my decisions at all stages.
- I can explain how I would approach the hard elements differently next time (ie what I would do differently).
- I will use this as an opportunity to try new techniques and improve my problem solving.

Further reading and resources

Banga, C. and Weinhold, J. (2014). *Essential Mobile Interaction Design*. New Jersey: Addison Wesley.

Lee, W-M. (2012). *Beginning Android 4 Application Development*. New Jersey: John Wiley & Sons.

Burd, B. (2015). *Android Application Development All-in-One For Dummies*. New Jersey: John Wiley & Sons.

Lowe, D. (2014). *Java All-in-One For Dummies*. New Jersey: John Wiley & Sons.

Ray, J. (2015). *iOS 8 Application Development in 24 Hours, Sams Teach Yourself*. Indianapolis: Sams Publishing.

Websites

Develop apps for Android™: <http://developer.android.com/>

This enables you to get started with Android Studio.

Develop apps for Apple®: <https://developer.apple.com/programs/>

The Apple Developer Program.

Mobile Development Tutorials: www.tutorialspoint.com/

Mobile Development Made Easy (for Android™, Apple® iOS etc.).

THINK ▶ FUTURE



Elizabeth Shaw

Junior Developer
in a mobile app
development
team

I've been working on mobile apps for about two years and during this time I have been involved in developing many different apps, some for Android™ and others for Apple® devices. Although the technologies and languages are quite different, the design principles are generally very similar. Although sometimes my preferred design is pitch-perfect, this often isn't the case and I have to review the design with considerable input from my colleagues. I have to be mature about this and take suggestions and feedback in a positive way although often it will mean going back to the drawing board and trying other ideas.

Of course, sometimes I don't agree with my colleagues and I have to justify the decisions I have made during the design process. As long as I can prove that my ideas meet the identified requirements this is usually ok. Essentially, I am demonstrating my creativity and taking an individual responsibility for my efforts, something which my colleagues openly encourage.

Focusing your skills

Reviewing mobile app designs

It is important to be able to review the designs of a mobile app, both individually and with others, in order to ensure that they meet the identified requirements. Here are some questions to ask yourself to help you do this.

- What are the implications of implementing a poorly thought-out design?
- Which requirements need to be identified and addressed by your design?
- How can you justify the decisions that you made during the design process?
- How you would record the feedback you receive from others and how might you use it to improve the design?
- Finally, for each requirement you have identified, work out how you can ensure that it is dealt with in your design and review.

Designing a mobile app – questions on design requirements

The design for your mobile app has been reviewed by others. What happens next?

- If the mobile app design is poorly received and serious shortcomings have been identified in your understanding of the problem and/or your design, you will need to revisit the original client requirements.
- If the mobile app design is generally well received but issues have been identified with elements of your designs that rely on specific devices, platforms or the presence of integrated hardware, you may wish to refine your ideas.
- If the mobile app design is very well received, consider that there may still be room for improvement and seek advice and guidance from others who may have more practical experience than you.

Getting ready for assessment



Florence is working towards a BTEC National in Information Technology. She was given an assignment which asked her to give a presentation on 'mobile apps and mobile devices' for Learning Aim A. She had to cover all the different areas of mobile apps and mobile devices, with a particular emphasis on which aspects affect the design and implementation of mobile apps. Florence shares her experience of completing this assignment below.

How I got started

Having researched the different types of mobile app (including the different categories of app and whether they are native, hybrid or web apps), my next step was to think about how the users' needs determined the app's design. To do this, I asked a relative for an app that they would like on their mobile phone. I made notes of their needs (what they wanted it to actually do), their preferences (how they wanted it to look and work) and their characteristics (how they wanted to use it) and tried to link this to a potential design, explaining my decisions in terms of their requirements.

Another aspect I had to explore was how current technologies impacted on the design and implementation of mobile apps. I looked back at apps operating on various devices (for size comparison) and from differently aged models (older and newer phones).

I tried to analyse this further by selecting a number of different apps and comparing their design and implementation across various types and models of device. I managed to create a list of different aspects that enabled me to compare and contrast these really well.

In order to achieve the highest grade, I realised I had to make some judgements about the effectiveness of these apps. I conducted and documented a consumer test with my family and friends, asking them to compare a popular app across a number of different devices. They gave me a lot of feedback about the usability of each app and I collated this into a set of results which I used to rate each app. It was then possible to draw some conclusions about the users' needs, the purpose of the app and the technologies available in each device in order to evaluate how the app's overall design and effectiveness had been directly affected.

How I brought it all together

I typed my findings into a document and used this to create slides and notes which formed my presentation. I created some tables and charts to help me to explain my findings. This reduced the amount of text I needed to add and made the presentation look more interesting and professional.

I ensured that I had proofread each page carefully and practised the presentation with a family friend who pretended to be the target audience. This helped with my nerves and timings on each slide. When they became confused over some of the technical terms I had used, I simply changed them or added a quick explanation. At the end of the report, I added a set of URLs for the audience so that they could find out more information.

What I learned from the experience

I realised that, although my class notes provided a good starting point, it was essential to understand what I was being asked to do before I started work on the assignment. This proved to be good practice for developing mobile apps: always understand the problem fully before you start work on it.

I also learned that collecting other people's opinions was a very useful way of getting honest feedback and this helped me to rethink my own views on certain apps and mobile devices. Even when I didn't particularly agree with them, I still found their feedback interesting, something which will be useful when I receive feedback on my own app.

Think about it

- ▶ Do you know the difference between explaining, analysing and evaluating?
- ▶ Can you respect other people's feedback even when you do not necessarily agree with it?